Bash Regex Notes

1. What is Regex?

Regex (Regular Expression) is a sequence of characters that define a search pattern. It's used for:

- Searching text (`grep`, `sed`, `awk`)
- Validating input (emails, phone numbers)
- Replacing text
- Splitting strings

In Bash, regex is often used with:

- `[[ string =~ regex ]]` (Bash built-in)
- `grep`, `egrep`, `sed`, `awk`

---

2. Regex in Bash Conditional Expressions

Bash supports regex using the `[[ ... =~ ... ]]` syntax.

```
#!/bin/bash

string="hello123"

if [[ $string =~ [0-9]+ ]]; then
  echo "Contains numbers"
else
  echo "No numbers"
fi
```

Explanation:

- `[[ ... ]]` → Bash conditional expression
- `=~` → Regex matching operator
- `[0-9]+` → One or more digits

Important points:

- Regex in Bash is extended regex by default (no `-E` needed)
- Use quotes carefully; quoting the regex may break it
- Parentheses `()` need `\(` and `\)` if not using extended regex in `grep`

---

3. Regex Metacharacters

Symbol	Meaning	Example	Matches
`.`	Any single character	`a.c`	`abc`, `axc`, `a-c`
`*`	Zero or more of previous	`lo*l`	`ll`, `lol`, `lool`
`+`	One or more of previous	`lo+l`	`lol`, `lool`
`?`	Zero or one	`colou?r`	`color`, `colour`
`^`	Start of string	`^hello`	`hello world`
`$`	End of string	`world$`	`hello world`
`[]`	Character class	`[aeiou]`	Any vowel
`[^]`	Negated class	`[^0-9]`	Any non-digit
`|`	OR	`|cat`	
`()`	Grouping	`(ab)+`	`ab`, `abab`
`\`	Escape special chars	`\.`	Matches `.` literally

Regex Shorthand Character Classes

Regular expressions provide shorthand character classes to simplify matching common types of characters. Here's a detailed reference:

Symbol	Meaning	Example
`\d`	Digit (`[0-9]`)	`\d\d` → matches `23`, `99`
`\D`	Non-digit	`\D+` → matches `"abc"`, `"!!"`
`\w`	Word character (`[A-Za-z0-9_]`)	`\w+` → `"Hello_123"`
`\W`	Non-word character	`\W+` → `"!!"`, `" "`
`\s`	Whitespace (space, tab, newline)	`\s+` → `" "`
`\S`	Non-whitespace	`\S+` → `"word"`

Explanation of Each Symbol

`\d`: Matches any single digit from `0` to `9`.

- Example: `\d\d` matches `"23"`, `"99"` in a string.

`\D`: Matches any character that is **not** a digit.

- Example: `\D+` matches `"abc"` in `"abc123"`.

`\w`: Matches any word character (letters, digits, or underscore). Equivalent to `[A-Za-z0-9_]`.

Example: `\w+` matches `"Hello_123"`.

`\W`: Matches any character that is **not** a word character.

- Example: `\W+` matches punctuation or spaces, like `"!!"` or `" "`.

`\s`: Matches any whitespace character (space, tab, newline).

- Example: `\s+` matches multiple spaces `" "`.

`\S`: Matches any character that is **not** whitespace.

- Example: `\S+` matches `"word"` in `" word "`.

Notes

- These shorthand classes are widely supported in most regex engines, including **Bash**, **Python**, **JavaScript**, **PHP**, and **Perl**.
- They are extremely useful for validating patterns like phone numbers, usernames, or parsing text efficiently.
- Combining them with quantifiers (`+`, `*`, `{m,n}`) allows flexible pattern matching.

Example in Bash:

```
text="User123 !!"
if [[ $text =~ \w+ ]]; then
  echo "First word: ${BASH_REMATCH[0]}"
fi



## 4. Common Bash Regex Examples

### 4.1 Validate a number
```bash
num="1234"
if [[ $num =~ ^[0-9]+$ ]]; then
```
```

```
  echo "Valid number"
fi
```

4.2 Validate an email

```
email="test@example.com"
regex="^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
if [[ $email =~ $regex ]]; then
  echo "Valid email"
else
  echo "Invalid email"
fi
```

4.3 Extract part of a string

```
text="User: John, Age: 25"
if [[ $text =~ Age:\ ([0-9]+) ]]; then
  echo "Age is ${BASH_REMATCH[1]}"
fi
```

- `BASH_REMATCH[0]` → full match
- `BASH_REMATCH[1]` → first captured group

4.4 Using grep with regex

```
echo "apple 123" | grep -E '[0-9]+'
```

- `-E` → Extended regex
- Outputs lines matching regex

---

5. Special Bash Regex Tips

- No quotes around regex in `[[ ... =~ ... ]]` unless necessary
- Spaces matter: `[0-9]+` ≠ `[0-9] +`
- Repetition `{min,max}`:

```
[[ "aaab" =~ a{2,4}b ]] && echo "Matches"
```

- Case-insensitive with `grep`:

```
echo "Hello" | grep -i 'hello'
```

- Negate regex:

```
if [[ ! $string =~ [0-9] ]]; then
  echo "No digits"
fi
```

---

6. Regex with sed

```
text="My number is 1234"
echo $text | sed -E 's/[0-9]+/5678/'
# Output: My number is 5678
```

---

7. Regex with awk

```
echo -e "apple\nbanana\ncherry" | awk '/a/ {print $0}'
# Output:
# apple
# banana
```

- Awk uses regex for pattern matching by default.

---

8. Capturing Groups in Bash

```
str="name:John age:30"
if [[ $str =~ name:([A-Za-z]+)\ age:([0-9]+) ]]; then
  echo "Name: ${BASH_REMATCH[1]}"
  echo "Age: ${BASH_REMATCH[2]}"
fi
```

---

9. Summary

- Bash supports extended regex with `[[ ... =~ ... ]]`
- Use `BASH_REMATCH` for captured groups
- Common tools (`grep`, `sed`, `awk`) also use regex
- Regex is powerful for validation, extraction, and transformation
- Escape special characters if needed (`\.` for dot, `\+` for plus)

Bash Regex Cheat Sheet

1. Bash Regex Basics

```
string ='hello123
if [[ String == regex ] ] hen
    Match found
$[BASH_REMATCH[0]
$[BASH_REMATCH[1]
```

3. Common Bash Regex Examples

Numbers

```
[["ʃ234" == "0-9+$ ] ]] & eao Number"
```

Letters Only

```
[[Hello" == "A-ta-zi+$]] & eoe "Letters"
```

Alphanumeric

```
[['User125" =="Axta z0-9/+$ & ecoe "Vaitøil"
```

Emall Validation

```
regex="{a-zA-Z-0 .9 ..)$ł-][@Ja·z A-Z-9]
$ )\a.z "balid email"
```

URL Validation

```
[[ 'https://example.com' == "httos?·//
[a-z-A-Z-8 -,l), [▤-zA-2](2,) (/:+)?3]
&& ec/bc 'Valid URL"
```

5. Regex Operators

```
e<o  E = [' Fi/ == li l*D
! =x ||= "abc" == "O-9]
&& echo Matches 2-4 a's followed by
```

2. Core Regex Symbols

Symbol	Meaning	Example	Mathes
ʘ.	Any ch	a + c	c ∤c
*	Laper	0-1 ʃ	(, Q
?	Orgaɹe	e elt	a ¯ q
0	Use	a e e	u eo
$	Grouping	(+))	N T a
/	R.and	1 + +	2 + o
‡	Q ver	b e ਖ	a c1
()	(0)	ε l+ ‡	c o
[ ]	ʄ ˀ e q	3 a ɘ	a a e

4. Regex Operators

==	Matches regex
! =-	Does NOT match regex

```
[[ [ 'abc' == "≣ 0-9] ]] & echo " No
```

7. Advanced Examples

Password Validation

```
[[ "MyPass123" == "(?;+!(a-z)(?;?]}
(?; 'ʎ 2))$,∂e & echo 'Strong password"
```

Repetitions

```
[[ "anab" == a(2,4} b]] & echo
'Matches 2-4 a's followed by b'
```

Negating

```
[[ t 'hello' == [0-9] ]] & echo
"No digits'
```